Gated networks: an inventory

Olivier Sigaud, Clément Masson, David Filliat, Freek Stulp

GATED NETWORKS: AN INVENTORY

Olivier Sigaud
Sorbonne Universités, UPMC Univ Paris 06, UMR 7222, F-75005 Paris, France
CNRS, Institut des Systèmes Intelligents et de Robotique UMR7222, Paris, France
olivier.sigaud@isir.upmc.fr +33 (0) 1 44 27 88 53

Clément Masson, David Filliat, Freek Stulp
École Nationale Supérieure de Techniques Avancées (ENSTA-ParisTech)
FLOWERS Research Team, INRIA Bordeaux Sud-Ouest.
828, Boulevard des Maréchaux, 91762 Palaiseau Cedex, France
{clement.masson,david.filliat,freek.stulp}@ensta-paristech.fr

ABSTRACT

Gated networks are networks that contain gating connections, in which the outputs of at least two neurons are multiplied. Initially, gated networks were used to learn relationships between two input sources, such as pixels from two images. More recently, they have been applied to learning activity recognition or multimodal representations. The aims of this paper are threefold: 1) to explain the basic computations in gated networks to the non-expert, while adopting a standpoint that insists on their symmetric nature. 2) to serve as a quick reference guide to the recent literature, by providing an inventory of applications of these networks, as well as recent extensions to the basic architecture. 3) to suggest future research directions and applications.

1 INTRODUCTION

Due to its many successful applications to pattern recognition, deep learning has become one of the most active research trends in the machine learning community (LeCun et al., 2015). The main building blocks in the deep learning literature are Restricted Boltzmann Machines (RBMs) (Smolensky, 1986), autoencoders (Hinton & Salakhutdinov, 2006; Vincent et al., 2008), Convolutional Neural Networks (CNNs) (LeCun et al., 1998) and Recurrent Neural Networks (RNNs) (Bengio, 2013).

Most of these architectures are used to learn a relationship between a single input source and the corresponding output. They do so by building a representation of the input domain that facilitates the extraction of the adequate relationship. However, there are many domains where the representation to be learned should relate more than one source of input to the output.

In reinforcement learning, for instance, value functions take a state and an action as input, and output a expected return. In order to deal with continuous states and actions, finding separately the adequate representations for states and actions to facilitate value function learning might be critical (Mnih et al., 2015; Lillicrap et al., 2015). Moreover, there are cases where learning a *reversible* tripartite relationship might be particularly useful. For instance, in control problems, forward models take a state and an action as input, and output the next state whereas inverse models take the current state and a desired state as input, and output an action. It would be interesting to learn a single representation for both models which could be used both in the forward and the inverse way.

Gated networks are extensions of the above deep learning building blocks that are designed to learn relationships between at least two sources of input and at least one output. A defining feature of these architectures is that they contain *gating connections*, as visualized in Figure 1. When the relationships between several sources of data involves multiplicative interactions, such gating connections between neurons result in more natural topologies and increase the expressive power of neural networks, because implementing a multiplicative relationship between two layers of stan-

dard neurons would require a number of dedicated neurons that would grow exponentially with the required precision (Memisevic, 2013).

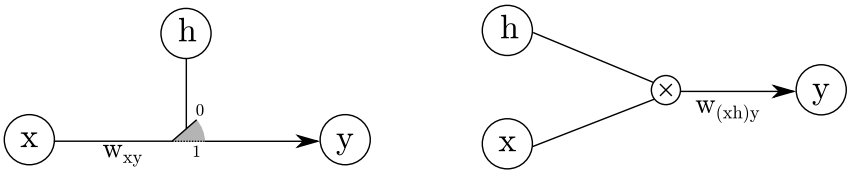


Figure 1: Two types of gating connections. On the left hand-side, the h neuron acts as a switch or gate that stops or not the flow of information between x and y. On the right hand-side, the connection implements a multiplicative relationships between the inputs x and h to provide the output y. Image reproduced from (Droniou, 2015).

Although the history of gating connections dates back at least to 1981 (Memisevic, 2013), there has been a recent surge of interest in these networks. Initially they were mainly used to learn transformation between images (Memisevic & Hinton, 2007), but they have recently also been applied to human activity recognition from videos and moving skeleton data from the kinect sensor (Mocanu et al., 2015), or to recognize offspring relationship from pictures of faces (Dehghan et al., 2014).

In robotics, gated networks have been used to learn to write numbers (Droniou et al., 2014), as well as to learn multimodal representations of numbers using images, vocal signal and articular movements with the iCub robot (Droniou et al., 2015). At a higher level, the same tools could be used to learn affordances, that are often represented as object-action-effect complexes (Montesano et al., 2008). All these examples have led to the claim that gated networks might be a particularly suitable tool along the way towards *deep developmental robotics* (Sigaud & Droniou, 2016, to appear).

Despite this growing interest, the literature about gated networks is still sparse enough so that it can be covered into a short survey. The aims of this paper are to cover the basics of gated networks for the non-expert, to serve as an inventory of applications of gated networks, and to suggest future research directions and applications.

The rest of this paper is structured as followed. In the next section, we give a detailed account of the calculations performed by the standard gated network model and a few variants whose relationship to the standard model is highlighted after some generalization. In this presentation, we emphasize the symmetric nature of these networks because it reveals the connections between some of the surveyed works. Then we present the standard unsupervised learning mechanisms that are used for tuning these networks, and provide an inventory of the various uses which is summarized into a table. Finally, we survey a few recent architectures that include the core ingredient of gated networks, and conclude with directions for future research.

2 Standard Gated Network Architectures

Gated networks are networks where the input of some computational units (or "neurons") is a function of the product of the output of several other neurons. As illustrated in Figure 1, one can consider two kinds of connections between 3 neurons. In the first family, a neuron h is used as a switch that stops or not the flow of information between two other neurons x and y. This functionality is very similar to that of transistors as electronic switches in digital circuits. This mechanism is used in the LSTM family of networks (Hochreiter & Schmidhuber, 1997a; Srivastava et al., 2015), among others. In the second family, the gating connection implements a multiplicative relationship between two inputs x and y. Note that the latter mechanism is more general than the former, since a value of 0 in h gates y to 0. The most general view is that neuron h modulates the signal between x and y.

In this paper, we focus on the specific family of neural networks implementing a multiplicative relationship that are built on RBMs and autoencoders and, to a lesser extent, on CNNs and RNNs.

2.1 FROM GATED RBMS TO GATED AUTOENCODERS AND BEYOND

We now briefly introduce Restricted Boltzmann Machines (RBMs) (Smolensky, 1986), autoencoders (Hinton & Salakhutdinov, 2006; Vincent et al., 2008), Convolutional Neural Networks (CNNs) (LeCun et al., 1998) and Recurrent Neural Networks (RNNs) (Bengio, 2013), and show how these networks have been extended to contain gated connections.

An RBM is not a neural network but a particular probabilistic graphical model (PGM) (Koller & Friedman, 2009) whose graph is bipartite: one set (or layer) of nodes is called "visible" and is used as the input of the model, whereas the other layer is "hidden" and is interpreted as being the hidden cause explaining the input. Both layers are generally binary (though it is possible to extend them to real-valued units) and fully connected to each other. However, there are no connections within a layer, which facilitates inference and training. Training an RBM consists in finding the parameters (edge's weights and node's bias) which maximize the likelihood of the training data. Importantly, RBMs are *generative* models: they can model the probability density of the joint distribution of visible and hidden units, which enables them to generate samples similar to those of the training data onto the visible layer.

The first instance of a gated network in the deep learning literature was a gated RBM (GRBM) (Memisevic & Hinton, 2007). However, this model was using a fully connected multiplicative network that required a lot of memory and computations for inference and training. In the next section, we present a solution to this issue, that was introduced by Memisevic & Hinton (2010) as a direct extension of (Memisevic & Hinton, 2007), still using GRBMs.

Autoencoders also contain an input and a representation layer but, in contrast to RBMs, they are deterministic models. They are trained to encode the input into the latent representation layer and then to reconstruct (or decode) the input from that representation. In their basic form, they are *discriminative* models, which can only compute the hidden layer given an input. It was then shown that a particular class of regularized autoencoder, the denoising autoencoder (DAE), could learn a model of the data generating distribution. This endow autoencoders with generative properties similar to those of RBMs (Vincent et al., 2008). More formally, a DAE can be interpreted as a Gaussian RBM (Vincent, 2011).

This led to a shift from GRBMs to gated autoencoders (GAEs) (Memisevic, 2008; 2011; 2012) though research on GRBMs is still active (Taylor et al., 2010; Ding & Taylor, 2014).

Convolutional Neural Networks are an early family of deep learning architectures which are composed of alternating convolutional layers and pooling layers. They are inspired from the human vision system and they proved particularly efficient for image processing applications. Finally, RNNs contain at least one recurrent connection, which makes them adequate for dealing with temporally extended information (Hochreiter & Schmidhuber, 1997b).

The gating idea was also applied to RNNs (Sutskever et al., 2011) and CNNs, either combined to GRBMs (Taylor et al., 2010) or to GAEs (Konda & Memisevic, 2015), as we outline in Section 5.

2.2 REDUCING THE NUMBER OF MULTIPLICATIVE CONNECTIONS

Implementing a gated network requires memory. Consider the network shown in Figure 2(a), consisting of three layers $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{h}$[1] whose respective cardinality is n_x, n_y and n_h. Predicting the output layer $\hat{\mathbf{y}}$ given $\mathbf{x}$ and $\mathbf{h}$ with such a multiplicative network consists in computing all the values $\hat{y}_j$ of $\hat{\mathbf{y}}$ using

$$\forall j, \hat{y}_j = \sigma_y(\sum_{i=1}^{n_x}\sum_{k=1}^{n_h} W_{ijk}x_i h_k) \tag{1}$$

where σ_y is some (optional) non-linear *activation* function described in more details in Section 2.4.

Alternatively, one may compute $\hat{\mathbf{x}}$ given $\mathbf{y}$ and $\mathbf{h}$ or compute $\hat{\mathbf{h}}$ given $\mathbf{x}$ and $\mathbf{y}$ using

[1]Throughout this document, bold lowercase symbols denote vectors, and bold uppercase symbols denote matrices.

$$\forall i, \hat{x_i} = \sigma_x(\sum_{j=1}^{n_y}\sum_{k=1}^{n_h} W_{ijk} y_j h_k), \qquad \forall k, \hat{h_k} = \sigma_h(\sum_{i=1}^{n_x}\sum_{j=1}^{n_y} W_{ijk} x_i y_j).$$

Regardless of the σ functions, these models are called *bilinear* because, if one input is held fixed, the output is linear in the other input.

The weights W_{ijk} define a *3-way tensor*, which is used to compute $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$ or $\hat{\mathbf{h}}$ given both other vectors. This tensor contains $n_x \times n_y \times n_h$ connections. If n_x, n_y and n_h are in the same order of magnitude, the number of weights (aka parameters) is cubic in this magnitude.

Factored architectures are designed to avoid representing this cubic number of weights. Two ways to reduce this memory requirement are:

- Projecting the input and output, potentially high-dimensional signals, into a smaller space through *factor layers*, and then performing the central product between these smaller dimensions.
- Constraining the structure of the global 3-way tensor so as to restrict the number of weights.

In the next two sections, we show that the standard gated network takes the best of both views, by setting a constraint on the 3-way tensor structure that implements a projection onto factor layers, but that also avoids representing the full central product. Another striking feature of this architecture is that the resulting central product does not contain any tunable parameter.

2.2.1 PROJECTING ONTO FACTOR LAYERS

One way of reducing the number of weights consists in projecting the $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{h}$ layers onto smaller layers noted respectively $\mathbf{f}^x$, $\mathbf{f}^y$ and $\mathbf{f}^h$ before performing the product between these smaller layers. Given their multiplicative role, these layers are called "factor" layers. The corresponding approach is illustrated in Figure 2(b). If the respective cardinality of the factors is n_{f_x}, n_{f_y} and n_{f_h}, the number of weights of the central 3-way tensor is $n_{f_x} \times n_{f_y} \times n_{f_h}$. To tune the whole network, additional weights must be added to this 3-way tensor, respectively $n_x \times n_{f_x}$, $n_y \times n_{f_y}$ and $n_h \times n_{f_h}$ for each layer, so the total number of weights is $(n_{f_x} \times n_{f_y} \times n_{f_h}) + (n_x \times n_{f_x}) + (n_y \times n_{f_y}) + (n_h \times n_{f_h})$.

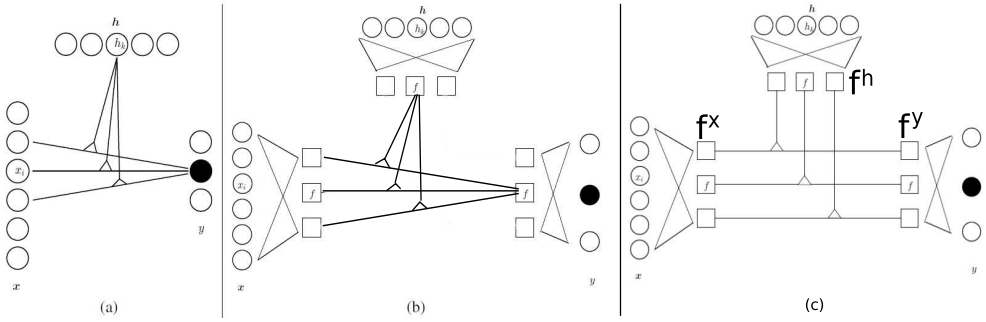


Figure 2: (a): A fully connected multiplicative network. (b): A simplified network introducing factor layers. (c): The factored gated architecture. All figures are adapted from (Memisevic et al., 2010).

In summary, the two input layers among $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{h}$ are first projected onto feature spaces through the corresponding factor layers, then the central 3-way multiplication is performed and finally projected to the output layer through the last factor layer.

This approach, suggested by Memisevic & Hinton (2007), results in fewer tunable parameters provided that factor layers contain fewer neurons than the input layers. In that case, the network performs *dimensionality reduction* on the inputs before tuning the multiplicative weights between the factors. As a result, the number of weights is still cubic, but of a smaller magnitude. A second benefit of this architecture is that, in contrast to the one illustrated in Figure 2(a), the introduction of factors results in the possibility of *feature sharing* between the different external layers (Memisevic et al.,

2010). However, to the best of our knowledge, this way of reducing the number of parameters of the gated architecture has not yet been implemented.

Another approach, which is used in all the works surveyed hereafter, consists in rather calling upon *over-complete* representations (Olshausen, 2003), where factor layers are larger than the input space, but a regularization method like *denoising* (Vincent, 2011) is used to sparsify the activity of the factors. In this context, introducing the factor layers does not reduce the number of parameters, it even increases it (Memisevic, 2013).

2.2.2 CONSTRAINING THE 3-WAY TENSOR

Another way of reducing the number of parameters consists in restricting the weights W_{ijk} to follow a specific form

$$W_{ijk} = \sum_{f=1}^{F} W_{if}^{x} W_{jf}^{y} W_{kf}^{h}. \tag{2}$$

With this constraint, the matrices $\mathbf{W}^x$, $\mathbf{W}^y$ and $\mathbf{W}^h$ are of respective size $n_x \times n_f$, $n_y \times n_f$ and $n_h \times n_f$, thus the total number of weights is just $n_f \times (n_x + n_y + n_z)$, which is quadratic instead of cubic in the size of input or factors.

Consider again the case where $\hat{\mathbf{y}}$ is predicted given $\mathbf{x}$ and $\mathbf{h}$. Equation (1) can be rewritten as

$$\forall j, \hat{y}_j = \sigma_y(\sum_{i=1}^{n_x}\sum_{k=1}^{n_h}\sum_{f=1}^{F} W_{if}^{x} W_{jf}^{y} W_{kf}^{h} x_i h_k), \tag{3}$$

which can be reorganized into

$$\forall j, \hat{y}_j = \sigma_y(\sum_{f=1}^{F} W_{jf}^{y}(\sum_{i=1}^{n_x} W_{if}^{x} x_i)(\sum_{k=1}^{n_h} W_{kf}^{h} h_k)). \tag{4}$$

By noting

$$f_f^x = \sum_{i=1}^{n_x} W_{if}^{x} x_i, \qquad f_f^y = (\sum_{j=1}^{n_y} W_{jf}^{y} y_j), \qquad f_f^h = (\sum_{k=1}^{n_h} W_{kf}^{h} h_k), \tag{5}$$

we finally get

$$\forall j, \hat{y}_j = \sigma_y(\sum_{f=1}^{F} W_{jf}^{y} f_f^x . f_f^h). \tag{6}$$

The three equations in (5) define three factor layers as explained in Section 2.2.1 and illustrated in Figure 2(b). However, when looking at the structure of (6), one can see that, instead of having a full central product, the output of both factor layers – $\mathbf{f}^x$ and $\mathbf{f}^h$ in the case of (6) – are multiplied element-wise through the same index f, as illustrated in Figure 2(c).

Thus, using the decomposition of (6), it can be seen that this way of constraining the 3-way tensor corresponds to using projections as in the previous view, but with three factor layers $\mathbf{f}^x$, $\mathbf{f}^y$ and $\mathbf{f}^h$ of the same size n_f, and where the central 3-way tensor has been replaced by 3 two-by-two element-wise products of the factor layers.

With a more algebraic notation, (5) can be rewritten

$$\mathbf{f}^x = \mathbf{W}^{x\intercal}\mathbf{x}, \qquad \mathbf{f}^y = \mathbf{W}^{y\intercal}\mathbf{y}, \qquad \mathbf{f}^h = \mathbf{W}^{h\intercal}\mathbf{h}. \tag{7}$$

In this notation, we omit the representation of an additive bias term by considering the inputs as being a homogeneous representation with an additional constant value, in which biases are implemented implicitly. Equation (6) then becomes

$$\hat{\mathbf{y}} = \sigma_y(\mathbf{W}^y(\mathbf{f}^x \otimes \mathbf{f}^h)), \tag{8}$$

where $\otimes$ denotes the element-wise multiplication illustrated in Figure 3(b).

Again, the same decomposition can be applied to predict $\hat{\mathbf{h}}$ given $\mathbf{x}$ and $\mathbf{y}$ or to predict $\hat{\mathbf{x}}$ given $\mathbf{y}$ and $\mathbf{h}$, giving rise to

$$\hat{\mathbf{x}} = \sigma_x(\mathbf{W}^x(\mathbf{f}^y \otimes \mathbf{f}^h)), \qquad \hat{\mathbf{h}} = \sigma_h(\mathbf{W}^h(\mathbf{f}^x \otimes \mathbf{f}^y)). \tag{9}$$

A slightly more general version of the same architecture that insists on its symmetric nature can be obtained by noting $\mathbf{W}^x_{\text{in}}$, $\mathbf{W}^y_{\text{in}}$ and $\mathbf{W}^h_{\text{in}}$ the matrices oriented from the input layers towards the factors, and $\mathbf{W}^x_{\text{out}}$, $\mathbf{W}^y_{\text{out}}$ and $\mathbf{W}^h_{\text{out}}$ those oriented from the factors towards the output. The corresponding architecture is depicted in Figure 3(b).

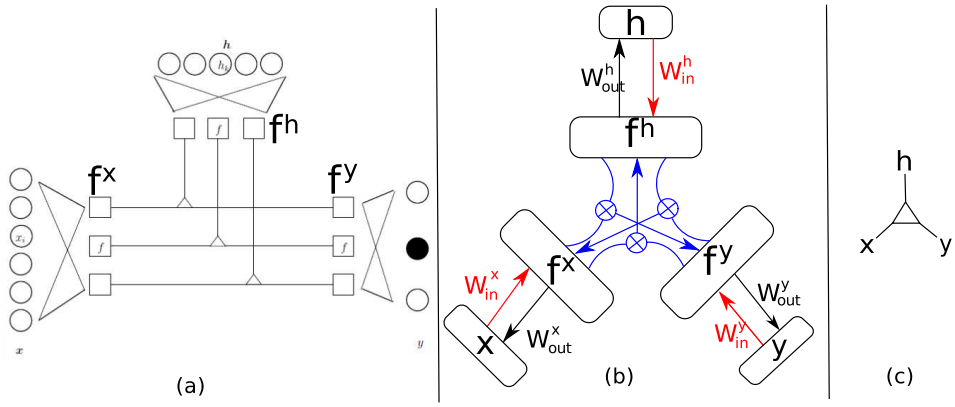


Figure 3: Two views of GAEs. (a): Same as Figure 2(c). (b): Another view of the same architecture, adapted from (Droniou, 2015). (c): Simplified notation corresponding to (b), reused in figures 5, 6 and 7.

Following these notations, if we consider computations from the input layers to the factors, the red arrows correspond to

$$\mathbf{f}^x_{\text{in}} = \mathbf{W}^x_{\text{in}}\mathbf{x}, \qquad \mathbf{f}^y_{\text{in}} = \mathbf{W}^y_{\text{in}}\mathbf{y}, \qquad \mathbf{f}^h_{\text{in}} = \mathbf{W}^h_{\text{in}}\mathbf{h}.$$

In the other way, from the factors to other factors, the blue arrows correspond to

$$\mathbf{f}^x_{\text{out}} = \mathbf{f}^y_{\text{in}} \otimes \mathbf{f}^h_{\text{in}}, \qquad \mathbf{f}^y_{\text{out}} = \mathbf{f}^x_{\text{in}} \otimes \mathbf{f}^h_{\text{in}}, \qquad \mathbf{f}^h_{\text{out}} = \mathbf{f}^x_{\text{in}} \otimes \mathbf{f}^y_{\text{in}}.$$

Finally, towards the output we have

$$\hat{\mathbf{x}} = \sigma_x(\mathbf{W}^x_{\text{out}}\mathbf{f}^x_{\text{out}}), \qquad \hat{\mathbf{y}} = \sigma_y(\mathbf{W}^y_{\text{out}}\mathbf{f}^y_{\text{out}}), \qquad \hat{\mathbf{h}} = \sigma_h(\mathbf{W}^h_{\text{out}}\mathbf{f}^h_{\text{out}}).$$

By connecting the above elements together, the complete input-output functions are

$$\hat{\mathbf{h}} = \mathbf{o}(\mathbf{x}, \mathbf{y}) = \sigma_h(\mathbf{W}^h_{\text{out}}((\mathbf{W}^x_{\text{in}}\mathbf{x}) \otimes (\mathbf{W}^y_{\text{in}}\mathbf{y}))), \tag{10}$$

$$\hat{\mathbf{x}} = \mathbf{p}(\mathbf{y}, \mathbf{h}) = \sigma_x(\mathbf{W}^x_{\text{out}}((\mathbf{W}^y_{\text{in}}\mathbf{y}) \otimes (\mathbf{W}^h_{\text{in}}\mathbf{h}))), \tag{11}$$

$$\hat{\mathbf{y}} = \mathbf{q}(\mathbf{x}, \mathbf{h}) = \sigma_y(\mathbf{W}^y_{\text{out}}((\mathbf{W}^x_{\text{in}}\mathbf{x}) \otimes (\mathbf{W}^h_{\text{in}}\mathbf{h}))). \tag{12}$$

Equations (10) to (12) are identical to (8) and (9), and thus they implement (2), provided that the following weight tying rules are used[2]: $\mathbf{W}^x = \mathbf{W}^x_{\text{in}} = \mathbf{W}^{x\,\intercal}_{\text{out}}$, $\mathbf{W}^y = \mathbf{W}^y_{\text{in}} = \mathbf{W}^{y\,\intercal}_{\text{out}}$ and $\mathbf{W}^h = \mathbf{W}^h_{\text{in}} = \mathbf{W}^{h\,\intercal}_{\text{out}}$. A benefit of using such weight tying rules is that it further reduces the number of parameters. Besides, any pair of the sub-networks described in (10) to (12) shares just one input matrix.

From the above presentation, it should be clear that the standard gated network architecture is completely symmetric: the functional role of the $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{h}$ layers can be exchanged without changing the computations.

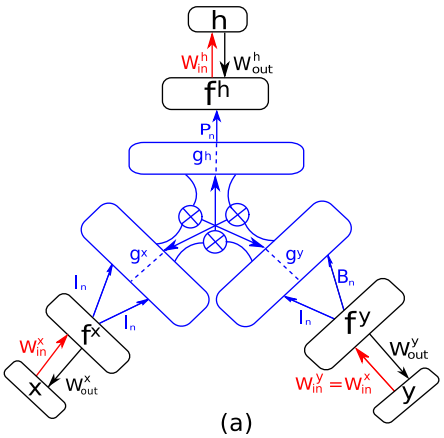
[2]Different papers choose different conventions for deciding which matrix is the original and which is the transposed, see for instance (Im & Taylor, 2014), giving rise to different equations to implement (10) to (12).

2.3 VARIATIONS ON THE CENTRAL TENSOR

The architecture outlined in Section 2.2.2 can be seen either as a particular way to parametrize the global 3-way tensor, introducing features into its internal structure, or as a way to replace the central tensor of the approach outlined in Section 2.2.1 by an element-wise product of factor layers. This approach to implementing the central 3-way tensor can be seen as a degenerate case where all its non-diagonal elements are null and its diagonal elements are all set to 1. With this definition, the central product does not contain any tunable parameters. Instead, representation learning is implemented by tuning the weights of the $\mathbf{W}^x$, $\mathbf{W}^y$ and $\mathbf{W}^h$ matrices connecting the external layers to the factors. Note that using parameters instead of ones onto the diagonal may increase the flexibility of the model for learning, but it would not improve its expressive power, since the effect of changing these parameters can be captured by changing the parameters of the $\mathbf{W}$ matrices.

The constraint given in (2) is somewhat arbitrary. For instance, the central computation of a gated architecture can be more complex than a simple element-wise product of factors. The architecture proposed in (Droniou & Sigaud, 2013) is an instance of such more complex computation. As outlined in Figure 4, it also uses factors and a parameter-free tensor, but the structure of the central tensor has been specifically designed to learn orthogonal transformations. Several motivations for performing the corresponding computations are given in (Droniou & Sigaud, 2013), together with the detailed mathematical rationale for such computations.

Note also that, in this architecture, the weight tying rules are unusual. Instead of having $\mathbf{W}_{\text{in}} = \mathbf{W}_{\text{out}}{}^{\intercal}$ for all factors, $\mathbf{W}^h_{\text{in}}$ and $\mathbf{W}^h_{\text{out}}$ are untied and $\mathbf{W}^x_{\text{in}} = \mathbf{W}^y_{\text{in}}$, with standard input-output weight tying rules on the $\mathbf{x}$ and $\mathbf{y}$ layers, i.e. $\mathbf{W}^x = \mathbf{W}^x_{\text{in}} = \mathbf{W}^x_{\text{out}}{}^{\intercal}$ and $\mathbf{W}^y = \mathbf{W}^y_{\text{in}} = \mathbf{W}^y_{\text{out}}{}^{\intercal}$. A consequence of this choice is that the model might not be interpreted as an energy-based dynamical system, since $\mathbf{W}^h_{\text{in}} = \mathbf{W}^h_{\text{out}}{}^{\intercal}$ is required so that Poincaré's integrability criterion holds (Im & Taylor, 2014).



$$\mathbf{B}_n = \begin{pmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} & 0 & 0 & 0 & 0 & \dots \\ 0 & \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} & 0 & 0 & 0 & \dots \\ 0 & 0 & \ddots & & & \end{pmatrix}.$$

$$\mathbf{P}_n = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 1 & 1 & 0 & 0 & \dots \\ & & & \ddots & & & \end{pmatrix}.$$

(b)

Figure 4: (a) In this architecture, an element-wise product is performed between vectors $\mathbf{g}^x$, $\mathbf{g}^y$ and $\mathbf{g}^h$ of size $2n$, where n is the size of $\mathbf{f}^x$, $\mathbf{f}^y$ and $\mathbf{f}^h$. The vector $\mathbf{g}^x$ is obtained by duplicating $\mathbf{f}^x$, using twice the identity matrix. One half of $\mathbf{g}^y$ is identical to $\mathbf{f}^y$, the other half is obtained through the block diagonal $\mathbf{B}_n$ matrix shown in (b). Finally, $\mathbf{g}^h$ is obtained from $\mathbf{f}^h$ by applying the $\mathbf{P}_n$ matrix. Note that the weight-tying rules differ from the ones of standard gated networks.

2.4 ACTIVATION FUNCTIONS

The $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{h}$ layers can be either binary of real-valued. Depending on this format, different non-linear *activation* functions are used, resulting in different functionalities assigned to the network.

When the content of a size n layer is binary, there are two options. First, it can represent 2^n elements of a discrete set using standard binary coding rules. In that case, either the model directly represents the probability of each binary value, as is the case in RBMs, or the binary values are obtained from real-valued numbers by using a threshold. This latter case is uncommon because of the non-differentiability of the threshold function. The second option is to represent only n ele-

ments using a "one-hot" representation where one bit is set to 1 and all the others are 0. When the corresponding layer is used as input, this representation is easy to enforce from the external world.

For real-valued layers, the role of the activation function is to constrain the values of the output layer into a bounded domain, which can be obtained for instance with a sigmoid or a rectified linear unit, the latter being more popular in large network due to its faster computation. One can also use the *softplus* function σ_+, defined as $\sigma_+(x) = \log(1+\exp(x))$, which is a smooth version of the rectified linear unit (Glorot et al., 2011).

To get a representation that is close toa "one-hot" in a real-valued output layer, the activity of the most active neuron in that layer can be highlighted by using the *softmax* function. If used for instance on the $\mathbf{h}$ layer of a gated network, the *softmax* function is

$$\mathbf{h} = \sigma_{max}(\mathbf{W}^h_{\text{out}}\mathbf{f}_\mathbf{h}), \tag{13}$$

where

$$\sigma^i_{max}(\mathbf{h} = (h_1, \ldots, h_n)) = \frac{e^{h_i}}{\sum_j e^{h_j}}. \tag{14}$$

In addition to highlighting the most active neuron(s), this function makes sure that all activities sum to 1. Hereafter, we call the obtained representation a "soft one-hot".

Finally, if a binary "one-hot" representation was required as output, one could apply a postprocessing "winner-takes-all" function to a *softmax* layer, but we are not aware of any such use.

3 LEARNING IN GATED NETWORKS

Gated networks have two input layers and one output layer. One way to train such networks would be to use *supervised learning*: for a given pair of input layers, one would provide the expected output, and then train the network to minimize a function of the error between the expected and the obtained output. This is used in gated CNNs and gated RNNs (see Section 5). But GRBMs and GAEs are not trained in this way. Instead, the training process is designed to perform *unsupervised* learning, but differs between GRBMs and GAEs. In this paper, we do not cover training GRBMs, which is based on training RBMs. We refer the reader to (Swersky et al., 2010) for a clear presentation of the latter topic. Instead, we focus on training GAEs.

Given two input layers, GAEs are trained to *reconstruct* one of them. In order to explain this learning process, it is useful to recap how it is performed in autoencoders.

An autoencoder is composed of two functions:

- The encoding function that transforms the input vector $\mathbf{x}$ into a latent representation $\mathbf{h}$. A typical function is $\mathbf{h} = \mathbf{h}(\mathbf{x}) = \sigma_h(\mathbf{W}\mathbf{x} + \mathbf{b})$.
- The decoding function that reconstructs a representation $\hat{\mathbf{x}}$ of $\mathbf{x}$ from its latent representation $\mathbf{h}$. A typical function is $\hat{\mathbf{x}} = \mathbf{r}(\mathbf{h}) = \sigma_x(\mathbf{W}'\mathbf{h} + \mathbf{b}')$.

The cost function for autoencoders is generally related to the reconstruction error. This error is for instance the distance between $\mathbf{x}$ and $\hat{\mathbf{x}}$, typically the squared error $||\hat{\mathbf{x}} - \mathbf{x}||^2$. Learning then corresponds to applying an optimization algorithm such as a gradient-descent to the weights of the network so as to minimize this cost function. Thus, during training, the network learns the encoding function and the decoding function simultaneously, using $\hat{\mathbf{x}} = \sigma_x(\mathbf{W}'\sigma_h(\mathbf{W}\mathbf{x} + \mathbf{b}) + \mathbf{b}')$. The main outcome of this learning process is the generation of the latent representation $\mathbf{h}$, that must be informative enough about the input so as to allow its correct reconstruction.

To highlight the relationship between autoencoders and GAEs, we now consider that $\mathbf{h}$ is the latent representation and $\mathbf{x}$ and $\mathbf{y}$ are the input layers. Recalling (10) to (12), there are two ways to define a GAE as equivalent to an autoencoder. The encoding function is always $\mathbf{h} = \mathbf{o}(\mathbf{x}, \mathbf{y})$, while the decoding function can be either

$$\hat{\mathbf{x}} = \mathbf{p}(\mathbf{y}, \mathbf{h}) = \mathbf{p}(\mathbf{y}, \mathbf{o}(\mathbf{x}, \mathbf{y})) \tag{15}$$

or

$$\hat{\mathbf{y}} = \mathbf{q}(\mathbf{x}, \mathbf{h}) = \mathbf{q}(\mathbf{x}, \mathbf{o}(\mathbf{x}, \mathbf{y})). \tag{16}$$

Using (10) to (12), (15) can be rewritten

$$\hat{\mathbf{x}} = \sigma_x(\mathbf{W}^x_{\text{out}}((\mathbf{W}^y_{\text{in}}\mathbf{y}) \otimes (\mathbf{W}^h_{\text{in}}\sigma_h(\mathbf{W}^h_{\text{out}}((\mathbf{W}^x_{\text{in}}\mathbf{x}) \otimes (\mathbf{W}^y_{\text{in}}\mathbf{y})))))), \tag{17}$$

and (16) can be rewritten

$$\hat{\mathbf{y}} = \sigma_y(\mathbf{W}^y_{\text{out}}((\mathbf{W}^x_{\text{in}}\mathbf{x}) \otimes (\mathbf{W}^h_{\text{in}}\sigma_h(\mathbf{W}^h_{\text{out}}((\mathbf{W}^x_{\text{in}}\mathbf{x}) \otimes (\mathbf{W}^y_{\text{in}}\mathbf{y})))))). \tag{18}$$

One can note that $\mathbf{W}^y_{\text{out}}$ does not appear in (17) and $\mathbf{W}^x_{\text{out}}$ does not appear in (18), thus tuning these weights is not useful during training unless adequate weight tying rules are applied.

As outlined in Section 2.1, autoencoders can be endowed with properties similar to those of RBMs by using a denoising regularization function. There are three kinds of such functions, namely Gaussian noise, masking noise and salt and pepper noise (Rudy & Taylor, 2014). It is commonplace to apply to GAEs these regularization functions as they are to autoencoders. They are generally applied to all factor layers, but there are some exceptions. For instance, in (Rudy & Taylor, 2014), the denoising function is applied to $\mathbf{x}$ only.

Importantly, minimizing the squared reconstruction error of a DAE implements a regularized form of *score matching* (Vincent, 2011), which is itself a training criterion that favors the encoding of the manifolds where most of the input data is lying (Hyvärinen, 2005). The same applies to GAEs, but the nature of the represented manifolds depends on the encoded input-output relationships and on the format of the external layers. Besides, some other regularizations functions such as *dropout* (Srivastava et al., 2014) might also be applied to GAEs, but we are not aware of any work in this direction. For other practical hints on training gated networks, see also (Memisevic, 2013).

Finally, the back-propagation algorithm can perform gradient descent on the weights of some or all the implied $\mathbf{W}$ matrices.

Taken together, the reconstruction function, the regularization function and the learning rules define many different settings to learn representations with GAEs. We study other combinatorial aspects in the next section.

4 APPLICATIONS OF GATED NETWORKS

Given what we have presented so far, there are three respects in which the use of gated networks may vary. First, as outlined in Section 2.4 the content of the $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{h}$ layers is either binary, one-hot or real-valued. Second, as outlined in Section 3, gated networks can be trained in various ways using various training signals, regularization functions and cost functions. Third, different layers can be used either as input or output. All these variations give rise to different functional roles for the corresponding networks. The goal of this section is to make an inventory of such uses in the literature, which is finally summarized in Table 1.

4.1 FORMAT OF THE EXTERNAL LAYERS

In Section 2.4, we outlined the different activation functions that are used to deal with different format of the external layers. Here, we recapitulate the use of these formats in different models.

First, in all GRBMs, the $\mathbf{h}$ layer always uses standard binary encoding (Memisevic & Hinton, 2007; 2010).

Furthermore, most models use pixels of two images as $\mathbf{x}$ and $\mathbf{y}$ input. The transformation between these images stored in $\mathbf{h}$ is either binary (Memisevic & Hinton, 2007; 2010) or real-valued (Droniou & Sigaud, 2013; Dehghan et al., 2014). In both cases, what is learned is a manifold of the pixels in the $\mathbf{x}$ conditioned on those of the $\mathbf{y}$ layer, or *vice versa* (Memisevic & Hinton, 2007).

There are two models where the $\mathbf{y}$ layer is binary. First, the *gated softmax classification* model was used in the context of logistic regression, i.e. classification using a log-linear model, where

the output $\hat{\mathbf{y}}$ consisted of binary class labels, and the values of the $\mathbf{h}$ layer were also binary (Memisevic et al., 2010).

More recently, in the context of studying the generative property of GAEs, a model was proposed where the $\mathbf{y}$ input also consists of a class-conditional, one-hot representation, whereas $\mathbf{h}$ is a real-valued representation constrained by a rectified linear unit (Rudy & Taylor, 2014). The network is trained to regenerate examples from the MNIST and Toronto Faces Database images, thus $\mathbf{x}$ is a vector of pixels. In this context, the model represents class-conditional manifolds, i.e. a set of manifolds of the input data $\mathbf{x}$ with one manifold per corresponding class in $\mathbf{y}$. As the authors state, this use of the GAE "is akin to learning a separate DAE model for each class, but with significant weight sharing between the models. In this light, the gating acts as a means of modulating the model's weights depending on the class label" (Rudy & Taylor, 2014).

4.2 Training signal

As outlined in Section 3, GAEs can be trained to reconstruct either $\hat{\mathbf{x}}$ or $\hat{\mathbf{y}}$. When the input data is binary, the cross-entropy loss function is the default choice (Rudy & Taylor, 2014). When it is real-valued, the standard cost function is a squared reconstruction error. Therefore, when training to reconstruct $\hat{\mathbf{x}}$, it is $J = \frac{1}{2}||(\hat{\mathbf{x}}|\mathbf{y}) - \mathbf{x}||^2$, whereas for reconstructing $\hat{\mathbf{y}}$, it is $J = \frac{1}{2}||(\hat{\mathbf{y}}|\mathbf{x}) - \mathbf{y}||^2$.

The first option is the one chosen in (Rudy & Taylor, 2014). This makes the connection to autoencoders more explicit because they both take $\mathbf{x}$ as input and $\hat{\mathbf{x}}$ as output. But this contrasts with the rest of the literature, where it is more common to train to reconstruct $\hat{\mathbf{y}}$ (Memisevic & Hinton, 2007; 2010; Memisevic et al., 2010; Droniou & Sigaud, 2013; Michalski et al., 2014b;a).

A third option exists. If we want the model to be able to reconstruct $\hat{\mathbf{x}}$ given $\mathbf{y}$ and $\hat{\mathbf{y}}$ given $\mathbf{x}$ at the same time, we can use (Memisevic, 2011):

$$J = \frac{1}{2}||(\hat{\mathbf{x}}|\mathbf{y}) - \mathbf{x}||^2 + \frac{1}{2}||(\hat{\mathbf{y}}|\mathbf{x}) - \mathbf{y}||^2. \tag{19}$$

A particularly relevant case for using this symmetric signal is the case where $\mathbf{x} = \mathbf{y}$. In that case, the mapping units $\mathbf{h}$ learn covariances within $\mathbf{x}$ (Memisevic, 2011).

Interestingly, a model recognizing offspring relationship from pictures of faces combines generative and discriminative training, using two training signals (Dehghan et al., 2014). From one side, it learns a representation of the transformation between two faces using the symmetric cost function given in (19). But it also tries to determine offspring relationship as a binary representation, so it uses a softmax cost function during a supervised label learning process. Finally, both cost functions are combined into a weighted sum.

4.3 Input-Output function

We have outlined in Section 2.2.2 that the role of the $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{h}$ layers could be exchanged. This leads to three permutations where two layers among $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{h}$ are inputs, the third layer being the output. However, given the unsupervised training procedure described in Section 3, we see that, in addition to the three possibilities outlined above, one can also use it to predict either $\hat{\mathbf{x}}$ or $\hat{\mathbf{y}}$. Under this view, learning the latent representation $\mathbf{h}$ is a side effect, $\mathbf{h}$ being used neither as input nor as output, but being "reinjected" into the network to reconstruct one of the input layers. The same fact applies *mutatis mutandis* to all other layers.

The different possible output layers result in two main ways to use a gated network. The first one, the predictive coding view, consists in inferring an output $\hat{\mathbf{y}}$ (or $\hat{\mathbf{x}}$) given an input $\mathbf{x}$ and a context $\mathbf{h}$. The *temporal* predictive coding view is a special case of the above, with $\mathbf{x}_t$ as input and $\mathbf{x}_{t+1}$ as output. The second one, the *transformation coding view*, consists in using the latent representation $\mathbf{h}$ as output, given two input vectors $\mathbf{x}$ and $\mathbf{y}$. The output layer $\mathbf{h}$ then expresses some relations between $\mathbf{x}$ and $\mathbf{y}$, which may provide abstract representations that can be used for instance in higher level decision modules.

The latter view is mostly used to learn transformations between two successive images, so as to extract features containing temporal information (Memisevic & Hinton, 2007; 2010; Memisevic et al., 2010; Droniou & Sigaud, 2013; Michalski et al., 2014b;a). In this context, the input vectors $\mathbf{x}$ and

$\mathbf{y}$ are successive images, for instance from a video. The extracted transformations $\mathbf{h}$ are content-independent. For instance, they can represent rotations, independently from what is rotated in the images. Furthermore, they convey a temporal information about these successive images, thus they can be used as elementary features in a higher level to model some temporal information. However, we are not aware of any architecture where these temporal features are actually used to extract temporally extended information from videos, apart from very preliminary attempts among 3 or 4 successive frames in (Michalski et al., 2014b) using a hierarchical sequence of GAEs (see Section 5). The work of (Dehghan et al., 2014) is another instance of the transformation coding view, but where $\mathbf{x}$ and $\mathbf{y}$ are temporally independent images.

In many papers, both the transformation representation $\mathbf{h}$ and the reconstructed input signal $\hat{\mathbf{x}}$ or $\hat{\mathbf{y}}$ are studied. As a consequence, in the absence of an external architecture that uses it, it is often hard to determine which of these signals should be considered as the output of the network. Moreover, it is often the case that, when learning transformations between two successive images, the learned transformation is then applied to a new input image to see what output image is "fantasized" by the network, performing a type of "analogy making". In this context, the output of the network is both $\hat{\mathbf{h}}$ and $\hat{\mathbf{y}}$ (Memisevic & Hinton, 2007; 2010; Memisevic et al., 2010; Droniou & Sigaud, 2013; Michalski et al., 2014b;a). Thus in Table 1, we do not strive to determine which layer is the output of the studied algorithm.

4.3.1 SUMMARY: AN INVENTORY

Table 1 summarizes many uses of the standard gated networks listed above.

Papers	$\mathbf{x}$	$\mathbf{y}$	$\mathbf{h}$	act. func.	training
(Memisevic & Hinton, 2007; 2010)	pixels(t)	pixels(t+1)	binary	proba	$\hat{\mathbf{y}}$
(Memisevic et al., 2010)	pixels	binary	binary	proba	$\hat{\mathbf{y}}$
(Memisevic, 2011)	pixels	pixels = $\mathbf{x}$	soft 1-hot	relu	$(\hat{\mathbf{x}}, \hat{\mathbf{y}})$
(Droniou & Sigaud, 2013)	pixels(t)	pixels(t+1)	real	softplus	$\hat{\mathbf{y}}$
(Rudy & Taylor, 2014)	pixels	1-hot	real	relu	$\hat{\mathbf{x}}$
(Dehghan et al., 2014)	face 1	face 2	soft 1-hot	softmax	*hybrid*

Table 1: Various input-output functions for gated networks. "act. func" stands for the activation function on the $\mathbf{h}$ layer. "relu" stands for rectified linear unit, "real" stands for real-valued. "proba" stands for a probabilistic activation function. The $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ training signal stands for the symmetric cost function given in (19). For the *hybrid* training signal, see Section 4.2.

Table 1 illustrates that there is a wide variety of ways to use gated networks. This variety is even greater if we also consider the non-standard architectures surveyed in the next section.

5 BEYOND STANDARD GATED ARCHITECTURES

In this section, we describe a few architectures that contain a gated network. First, we list some architectures where the central tensor connects more than 3 layers. Then, we present some architectures whose set of connections is not restricted to the central tensor.

5.1 EXTENDED TENSORS

There are some architectures where the central tensor connects more than 3 external layers. Conditional RBMs (CRBMs) are RBMs where some memory of the past input are included into the input layer so that the architecture can model time-dependent data (Taylor & Hinton, 2009). In (Taylor et al., 2011), a CRBM is used to model human motion data but, as illustrated in Figure 5, it is extended with an additional *style* layer to model different styles of motion.

The $\mathbf{x}$ layer corresponds to the motion input at previous time step. The $\mathbf{y}$ layer, which is the output, corresponds to the predicted motion at the current time step. The $\mathbf{h}$ layer is used as in all GRBMs to learn a representation of the transformation between $\mathbf{x}$ and $\mathbf{y}$. But, additionally, the $\mathbf{z}$ layer

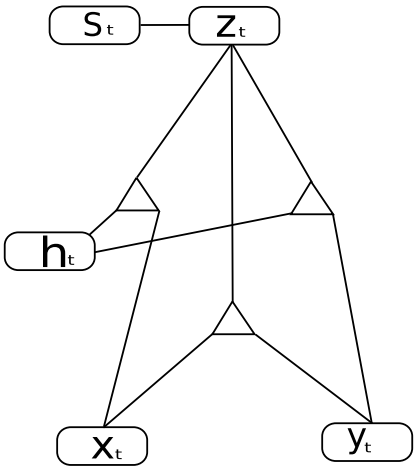


Figure 5: Four layers can be connected by three tripartite connection blocks (adapted from (Taylor et al., 2011)).

corresponds to real-valued stylistic features that are fed by discrete *style* labels (encoded in the $\mathbf{s}$ layer) and provide some additional *contextual* information about the motion.

The resulting architecture is then factored as described above so as to limit the number of parameters, but it is designed in such a way that the 4 factors are only connected together by triplets using factored 3-way tensors.

More recently, a "4-way tensor" and its factorization were introduced based in GRBMs (Mocanu et al., 2015). The central factored operation consists in performing a sum of products of second order tensors. The models of (Taylor & Hinton, 2009) and (Mocanu et al., 2015) are both capable of representing sequential data in the limit of the N previous time steps included in the memory concatenated to the input layer.

5.2 CLUSTERING WITH GATED NETWORKS

In some architectures, the central 3-way tensor is not the only ingredient. For instance, the architecture depicted in Figure 6 uses an additional autoencoding connection with respect to a standard GAE (Droniou et al., 2015).

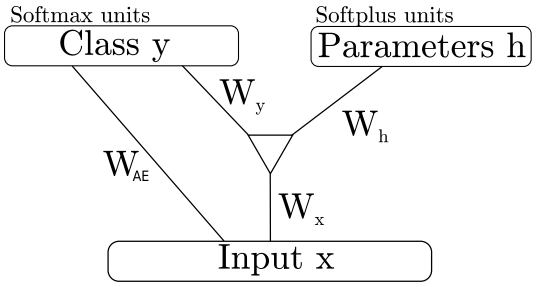


Figure 6: Gated network for unsupervised classification (adapted from (Droniou et al., 2015)). The GAE is represented using the simplified notation of Figure 3(c). With respect to a standard GAE, it uses an additional autoencoder implemented through the $\mathbf{W}_{AE}$ matrix.

The network aims at clustering input data into "concepts" corresponding to manifolds in the input layer, without using supervised learning. For doing so, the input $\mathbf{x}$ is first fed into a standard autoencoder using a softmax activation function that performs unsupervised clustering of the input data.

The softmax activation function implements a competition between the bits of the class layer and results in the emergence of a soft one-hot representation of the corresponding class. Then, given the input and the obtained class, the $\mathbf{h}$ layer implements a parametrization of the input with respect to the class, using a softplus layer, i.e. $\mathbf{h} = \sigma_+(\mathbf{W}^h_{\text{out}}\mathbf{f_h})$.

Since it uses a soft one-hot, class-conditional $\mathbf{y}$ layer and a real-valued input $\mathbf{x}$ layer, this model can be seen as a direct extension of the one presented in (Rudy & Taylor, 2014). However, since the weights are trained simultaneously, the network in Figure 6 finds the adequate classes to represent the data with an accurate parametrization by itself, instead of requiring them as training labels. This endows the network with unsupervised clustering capabilities that are well beyond those of standard dimensionality reduction techniques. This model is then extended to deal with multimodal information, showing an even improved clustering performance. We do not further study this aspect here, see (Droniou et al., 2015) for more details.

5.3 RECURRENT GATED NETWORKS

Another architecture based on factoring gating connections is the "Multiplicative RNN" architecture (Sutskever et al., 2011) depicted in Figure 7. This is a recurrent architecture trained to deal with temporally organized information such a text or speech signal.

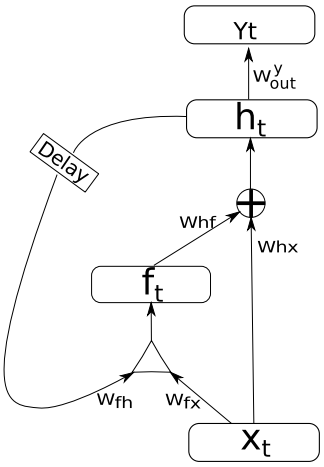


Figure 7: Multiplicative RNN.

The key requirement of the architecture is that the recurrent connection responsible for the dynamics of the hidden variable should be a function of the input layer $\mathbf{x}$. This would lead to a full 3-way tensor, which the authors factorize as described in Section 2.2 to reduce the number of free parameters. With slightly adapted notations to highlight the similarity with other architectures, the internal computation of the multiplicative RNN is given by the following equations:

$$f_t = \text{diag}(\mathbf{W}_{fx}\mathbf{x}_t).\mathbf{W}_{fh}\mathbf{h}_{t-1} \tag{20}$$

$$\mathbf{h}_t = \tanh(\mathbf{W}_{hf}f_t + W_{hx}\mathbf{x}_t) \tag{21}$$

$$\hat{\mathbf{y}}_t = \mathbf{W}^y_{\text{out}}\mathbf{h}_t + \mathbf{b}_y. \tag{22}$$

A key difference between this work and the other ones presented above is that the architecture is trained in a supervised way, rather than trained to reconstruct its input. The focus is thus not on extracting an abstract representation of the input. Another originality is that, instead of being trained with a standard first order gradient descent algorithm such as back-propagation, the architecture is trained using a second order method based on Hessian-free optimization (Martens, 2010). To our knowledge, despite its efficiency, no other gated network has been trained with this method.

5.4 CONVOLUTIONAL GATED NETWORKS

Convolution is a technique which consists in processing a large image by shifting a smaller filter to any position in the image and applying it over all positions. For instance, the same filter can be applied to recognize a pattern at any position in the image. Convolutional gated networks apply the convolution idea to gated networks. This has been done in GRBMs (Taylor et al., 2010) so as to extract spatio-temporal features in the context of human activity recognition, and in GAEs (Konda & Memisevic, 2015) to perform visual odometry from stereo pairs in a sequence of images captured from a moving camera.

5.5 PREDICTION WITH A SEQUENCE OF GATED NETWORKS

Another architecture models temporal data using a sequence of GAEs (Michalski et al., 2014b)[3]. Beyond a sequence, it even uses a hierarchy of GAEs to learn transformations of transformations. The model of (Michalski et al., 2014b), called Predictive Gating Pyramides (PGP), cascades two level of GAEs to predict sequences. As the authors state, the reconstruction error is inadequate in their context, thus the model is trained explictly to predict rather than to reconstruct. Actually, it is trained to predict over multiple steps. A strong assumption in PGP is that the highest-order relational structure in the sequence is constant. It uses Back-Propagation Through Time (BPTT) to perform gradient descent on the weights over time. However, the model is used to learn temporal features, it does not predict long sequences of images. And a major drawback is that the architecture requires as many GAEs as time steps.

6 CONCLUSION

In this paper, we have based our presentation of gated networks on a perspective that insists on their symmetric nature. Based on this particular perspective, we could highlight its richness by providing an inventory of the various ways they have been used so far in the literature. Given this richness, we believe standard gated networks still have a largely underexploited potential as a unifying tool for many domains where the relevant information is naturally expressed as tripartite relationships between three interdependent sources. Apart from the ones proposed in this paper, we hope many other application domains to gated networks will emerge in the next years.

Furthermore, as pointed out in Section 5, there are still rather few non-standard architectures based on the factored gating idea. We believe the list of such architectures will expand in the future, and also that gated networks should be included into more general frameworks that may contain several instances of such networks, as is already the case with (Michalski et al., 2014b) or (Droniou et al., 2014).

Finally, among other things, an interesting perspective to this work consists in combining it with the *contextual learning* perspective (Jonschkowski et al., 2015). Indeed, several contextual learning patterns might be implemented with gated networks, and some works about representation learning with gated networks might be interpreted in the framework of contextual learning.

ACKNOWLEDGMENTS

This work was supported by the European Union's Horizon 2020 research and innovation program within the DREAM project under grant agreement N^o 640891.

REFERENCES

Bengio, Yoshua. Deep learning of representations: Looking forward. In *Statistical language and speech processing*, pp. 1–37. Springer, 2013.

Dehghan, Afshin, Ortiz, Enrique G., Villegas, Ruben, and Shah, Mubarak. Who do I look like? determining parent-offspring resemblance via gated autoencoders. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1757–1764. IEEE, 2014.

[3](Michalski et al., 2014a) is the corresponding arXiv preprint

Ding, Weiguang and Taylor, Graham W. "Mental rotation" by optimizing transforming distance. *arXiv preprint arXiv:1406.3010*, 2014.

Droniou, Alain. *Apprentissage de représentations et robotique développementale: quelques apports de l'apprentissage profond pour la robotique autonome*. PhD thesis, UPMC-Paris 6, 2015.

Droniou, Alain and Sigaud, Olivier. Gated autoencoders with tied input weights. In *Proceedings of the 29th International Conference on Machine Learning*, volume 29, pp. 1–6, 2013.

Droniou, Alain, Ivaldi, Serena, and Sigaud, Olivier. Learning a repertoire of actions with deep neural networks. In *ICDL-Epirob*, pp. 1–6, 2014.

Droniou, Alain, Ivaldi, Serena, and Sigaud, Olivier. A deep unsupervised network for multimodal perception, representation and classification. *Robotics and Autonomous Systems*, 71:83–98, 2015.

Glorot, Xavier, Bordes, Antoine, and Bengio, Yoshua. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pp. 315–323, 2011.

Hinton, G. E. and Salakhutdinov, R. R. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786):504–507, 2006. ISSN 1095-9203.

Hochreiter, Sepp and Schmidhuber, Jürgen. LSTM can solve hard long time lag problems. In *Advances in Neural Information Processing Systems 9: Proceedings of the 1996 Conference*, volume 9, pp. 473. MIT Press, 1997a.

Hochreiter, Sepp and Schmidhuber, Jürgen. LSTM can solve hard long time lag problems. In *Advances in Neural Information Processing Systems 9: Proceedings of the 1996 Conference*, volume 9, pp. 473. MIT Press, 1997b.

Hyvärinen, Aapo. Estimation of non-normalized statistical models by score matching. In *Journal of Machine Learning Research*, pp. 695–709, 2005.

Im, Daniel Jiwoong and Taylor, Graham W. Analyzing the dynamics of gated auto-encoders in practice. *Unpublished manuscript?*, 2014.

Jonschkowski, Rico, Höfer, Sebastian, and Brock, Oliver. Contextual learning. *arXiv preprint arXiv:1511.06429*, 2015.

Koller, Daphne and Friedman, Nir. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

Konda, Kishore and Memisevic, Roland. Learning visual odometry with a convolutional network. In *International Conference on Computer Vision Theory and Applications*, 2015.

LeCun, Yann, Bottou, Léon, Bengio, Yoshua, and Haffner, Patrick. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

LeCun, Yann, Bengio, Joshua, and Hinton, Geoffrey E. Deep learning. *Nature*, 521(7553):436–444, 2015.

Lillicrap, Timothy P, Hunt, Jonathan J, Pritzel, Alexander, Heess, Nicolas, Erez, Tom, Tassa, Yuval, Silver, David, and Wierstra, Daan. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Martens, James. Deep learning via hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning*, pp. 735–742, 2010.

Memisevic, Roland. *Non-linear latent factor models for revealing structure in high-dimensional data*. PhD thesis, University of Toronto, 2008.

Memisevic, Roland. Gradient-based learning of higher-order image features. In *IEEE International Conference on Computer Vision (ICCV)*, pp. 1591–1598. IEEE, 2011.

Memisevic, Roland. On multi-view feature learning. In *Proceedings of the 28th Annual International Conference on Machine Learning*, pp. 1–8, 2012.

Memisevic, Roland. Learning to relate images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1829–1846, 2013.

Memisevic, Roland and Hinton, Geoffrey E. Unsupervised learning of image transformations. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, 2007.

Memisevic, Roland and Hinton, Geoffrey E. Learning to represent spatial transformations with factored higher-order boltzmann machines. *Neural Computation*, 22(6):1473–1492, 2010.

Memisevic, Roland, Zach, Christopher, Hinton, Geoffrey E., and Pollefeys, Marc. Gated softmax classification. In Lafferty, J, Williams, C. K. I., Shawe-Taylor, John, Zemel, R. S., and Culotta, A. (eds.), *Advances in Neural Information Processing Systems*, volume 23, pp. 1603–1611, 2010.

Michalski, Vincent, Memisevic, Roland, and Konda, Kishore. Modeling sequential data using higher-order relational features and predictive training. *arXiv preprint arXiv:1402.2333*, 2014a.

Michalski, Vincent, Memisevic, Roland, and Konda, Kishore. Modeling deep temporal dependencies with recurrent "grammar cells". In *Advances in neural information processing systems*, pp. 1925–1933, 2014b.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G., Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Mocanu, Decebal Constantin, Ammar, Haitham Bou, Lowet, Dietwig, Driessens, Kurt, Liotta, Antonio, Weiss, Gerhard, and Tuyls, Karl. Factored four way conditional restricted boltzmann machines for activity recognition. *Pattern Recognition Letters*, 2015.

Montesano, L., Lopes, M., Bernardino, A., and Santos-Victor, J. Learning object affordances: From sensory–motor coordination to imitation. *IEEE Transactions on Robotics*, 24(1):15–26, 2008.

Olshausen, Bruno A. Principles of image representation in visual cortex. *The visual neurosciences*, 2:1603–1615, 2003.

Rudy, Jan and Taylor, Graham W. Generative class-conditional autoencoders. *arXiv preprint arXiv:1412.7009*, 2014.

Sigaud, Olivier and Droniou, Alain. Towards deep developmental learning. *IEEE Transactions on Autonomous Mental Development*, 2016, to appear. doi: 10.1109/TAMD.2015.2496248.

Smolensky, Paul. Information processing in dynamical systems: foundations of harmony theory. In *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1*, pp. 194–281. MIT Press, 1986.

Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Srivastava, Nitish, Mansimov, Elman, and Salakhutdinov, Ruslan. Unsupervised learning of video representations using LSTMs. *arXiv preprint arXiv:1502.04681*, 2015.

Sutskever, Ilya, Martens, James, and Hinton, Geoffrey E. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 1017–1024, 2011.

Swersky, Kevin, Chen, Bo, Marlin, Benjamin, and De Freitas, Nando. A tutorial on stochastic approximation algorithms for training restricted boltzmann machines and deep belief nets. In *Information Theory and Applications Workshop (ITA), 2010*, pp. 1–10. IEEE, 2010.

Taylor, Graham W. and Hinton, Geoffrey E. Factored conditional restricted boltzmann machines for modeling motion style. In *Proceedings of the 26th annual international conference on machine learning*, pp. 1025–1032. ACM, ACM, 2009. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553505.

Taylor, Graham W., Fergus, Rob, LeCun, Yann, and Bregler, Christoph. Convolutional learning of spatio-temporal features. In *ECCV'10*, pp. 140–153, September 2010. ISBN 3-642-15566-9, 978-3-642-15566-6.

Taylor, Graham W., Hinton, Geoffrey E., and Roweis, Sam T. Two Distributed-State Models For Generating High-Dimensional Time Series. *The Journal of Machine Learning Research*, 12: 1025–1068, 2011. ISSN 1532-4435.

Vincent, Pascal. A connection between score matching and denoising autoencoders. *Neural computation*, 23(7):1661–1674, 2011.

Vincent, Pascal, Larochelle, Hugo, Bengio, Yoshua, and Manzagol, Pierre-Antoine. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103, New York, New York, USA, 2008. ACM Press. ISBN 9781605582054. doi: 10.1145/1390156.1390294.